МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

Є. Ю. БЕРКУНСЬКИЙ

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни "Web-технології та web-дизайн"

Рекомендовано Методичною радою НУК

Електронне видання
комбінованого використання на DVD-ROM

МИКОЛАЇВ • НУК • 2015

УДК 004.55(076)
ББК 73я73
Б 48

Укладач Є. Ю. Беркунський, старш. викладач

Рецензент К. В. Кошкін, д-р техн. наук, професор

Беркунський Є. Ю.

Б 48 Методичні вказівки до виконання лабораторних робіт з дисципліни "Web-технології та web-дизайн" / Є. Ю. Беркунський. – Миколаїв : НУК, 2015. – 25 с.

Наведено теоретичні відомості та довідкову інформацію для виконання завдань лабораторних робіт з дисципліни: основні відомості про структуру web-сайтів, стандартні елементи, конструкції мов HTML та CSS. Розглянуто основи створення динамічного контенту мовою Java.

Призначено для студентів денної і заочної форм навчання спеціальності 8.05010101 "Інформаційні управляючі системи та технології". Можуть бути використані студентами інших спеціальностей.

УДК 004.55(076)
ББК 73я73

Навчальне видання

БЕРКУНСЬКИЙ Євген Юрійович

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни "Web-технології та web-дизайн"

Комп'ютерне верстання *А. Й. Лихіна*
Коректор *М. О. Паненко*

Формат 60×84/16. Ум. друк. арк. 1,5. Обсяг даних 1160 кб. Тираж 15 прим.
Вид. № 51. Зам. № 69.

Видавець і виготівник Національний університет кораблебудування імені адмірала Макарова
просп. Героїв Сталінграда, 9, м. Миколаїв, 54025, e-mail : publishing@nuos.edu.ua

Свідоцтво суб'єкта видавничої справи ДК № 2506 від 25.05.2006 р.

ВСТУП

Сучасні web-технології – це комплекс технічних, комунікаційних, програмних методів розв'язання задач організації діяльності користувачів із застосуванням мережі Інтернет.

Web-технології – це концепція роботи з інформацією. Вона відрізняється такими особливостями:

1. Технічна основа Web-технологій – локальні та глобальні мережі, Інтернет
2. Застосування особливого типу тонких клієнтів: web-браузерів
3. Число споживачів інформації практично не обмежено.
4. Публікатор сам може задати особливі умови доступу до інформації.
5. В публикаціях можуть міститись посилання на інші публикації без обмеження на вміст та джерела матеріалів.
6. Активна робота пошукових машин.
7. Доставка та тиражування контенту практично безкоштовні.

Привабливість Web-технологій як засобу доставки інформації багато в чому визначає універсальний інтерфейс між людиною та комп'ютером, відносна простота підтримки веб-ресурсу, постійне зростання аудиторії Інтернету. Web-технології дозволяють створювати прості для опанування, легкодоступні, дуже дешеві, швидкооновлювані інформаційні, діалогові, довідкові системи.

Лабораторна робота № 1. Мова розмітки HTML

Завдання. Використовуючи мову розмітки HTML, створити шаблон WEB-сайту. Шаблон повинен складатися як мінімум з трьох сторінок: Титульна сторінка, Звичайна сторінка та Сторінка зворотного зв'язку.

Теоретичні відомості

HTML – це не мова програмування, а "мова розмітки" (HyperText Markup Language). Вона визначає зміст і структуру сторінки, але не зовнішній вигляд. Елементи мови мають структуру дерева (вкладені елементи), пропуски або ігноруються, або замінюються одним пропуском. Вузли дерева являють собою або текст (зміст), або "Структурні елементи", маркуються "тегами" і мають "атрибути".

```
<html>
  <head>
    <title>My first page</title>
  </head>
  <body>
    <p>Hello, World!</p>
  </body>
</html>
```

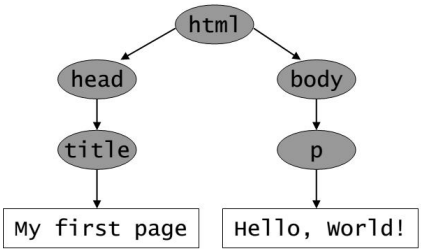


Рис. 1. Текст мовою HTML та його структура

Структура елементів мови HTML

Елемент, маркований тегом, має таку структуру:

```
<element attr1="value1" attr2="value2"...>
  внутрішній вміст
</element>
```

Наприклад, `<a href="page2.html">Next page</a>`

Якщо тег не має *внутрішнього вмісту*, то він може мати спрощену структуру:

```
<element attr1="value1" attr2="value2"... />
<img src="photo.jpg" alt="My photo" />
<hr/>
```

Структура сторінки

Правильна сторінка має дві частини – заголовок (інформація про сторінку) та тіло (вміст сторінки)

```
<html>
  <head>
    інформація про сторінку
  </head>
  <body>
    внутрішній вміст
  </body>
</html>
```

Браузери відображують сторінку, що прийшла до них по запиту. Сторінка може бути написана як на старому HTML, так і на нових XML та XHTML. Ми будемо використовувати більш сучасний XHTML.

Основна відмінність у внутрішньому вмісті документів HTML та XHTML: якщо браузер бачить помилку в HTML-документі, то він зобов'язаний спробувати зрозуміти, що мав на увазі автор документа. Якщо ж помилка виявлена у XHTML – документі, то браузер просто повідомляє про неї.

Крім того, форматування елементів XHTML-документа повинно бути оформлене за допомогою CSS-стилів. Існують теги, такі, як font, та атрибути, такі, як bgcolor і align, що не підтримуються в XHTML.

Вимоги XHTML, які дозволено порушувати у старому HTML

1. Всі елементи повинні бути закриті.
2. Всі обов'язкові атрибути повинні бути присутніми.
3. Вкладеність елементів повинна бути правильною.
4. В тілі документа текст не може бути вкладеним безпосередньо.
5. Атрибути повинні братися в лапки.
6. "Блочні" елементи не можуть бути вкладені в "рядкові".
7. Спецсимволи завжди повинні бути представлені мнемонічними посиланнями.
8. Теги та атрибути записуються лише рядковими літерами.

Блочні та рядкові елементи

Блочні елементи містять фрагменти тексту, які завжди відображуються в окремих блоках. Браузери візуально відокремлюють блочні елементи один від одного.

Приклади: `<div>, <p>, <ul>, <tr>`

Рядкові елементи можуть розташовуватись один за одним в межах одного рядка.

Приклади: `<img>, <a>, <b>, <span>`

Коментарі вставляються в HTML текст так же, як і звичайні елементи:

```
<!-- This is my coolest page -->
```

Деякі елементи HTML та їхні атрибути

Абзац (параграф) – блочний елемент

Так записується в редакторі:

```
<p>Видимий тут вміст відображується в браузері у вигляді одного абзацу.
Пропуски, що повторюються та переходи з рядка на рядок ігноруються.</p>
```

Так буде відображено:

Видимий тут вміст відображується в браузері у вигляді одного абзацу. Пропуски, що повторюються та переходи з рядка на рядок ігноруються.

Якщо необхідно зробити перехід на наступний рядок всередині абзацу, то це досягається за допомогою елемента

Так записується в редакторі:

```
<p>Це перший рядок параграфа,<br/>а це вже другий</p>
<p>А це вже наступний абзац.</p>
```

Так буде відображено:

Це перший рядок параграфа,
а це вже другий
А це вже наступний абзац.

Заголовки – блочні елементи

Так записується в редакторі:

```
<h1>Національний Університет Кораблебудування</h1>
<h2>Кафедра ІУСТ</h2>
<h3>Розклад занять</h3>
```

Так буде відображено:

Національний Університет Кораблебудування

Кафедра ІУСТ

Розклад занять

Всього може бути до шести рівнів заголовків (від <h1> до <h6>)

Горизонтальна риска (роздільник) – блочний елемент

Так записується в редакторі:

```
<p>Національний Університет Кораблебудування</p><hr/>
<p>Кафедра ІУСТ</p>
```

Так буде відображено:

Національний Університет Кораблебудування

Кафедра ІУСТ

Гіперпосилання – рядковий елемент

Важливу роль у web-сторінках відіграють гіперпосилання. Розрізняють посилання на інші сторінки та на позиції у поточному документі.

Так записується в редакторі:

```
<p>Користуйтеся пошуком
<a href = "http://www.google.com" > Google</a> -
найпоширенішою пошуковою системою у світі!</p>
```

Так буде відображено:

Користуйтеся пошуком Google – найпоширенішою пошуковою системою у світі!

Будьте акуратними із вкладеністю елементів!

```
<p><a href="Page2.html">Це в першому абзаці</p>
<p>А це вже у другому!</a></p>
```

Тут дві помилки: блочний елемент всередині рядкового і неправильна вкладеність елементів. Проте, HTML-браузер може правильно відобразити ці елементи!

Рисунки та фотографії

Вставка зображень – рядковий елемент.

```
<p><img src="server.jpg" alt="Сервер"/></p>
```

Додаткові атрибути

```
<img src="server.jpg" alt=" Сервер" width="250"
height="150" />
<img src="server.jpg" alt="Сервер" width="50%"/>
```

Зображення може служити посиланням так само, як і текст:

```
<a href="servers/server.html">
<img src="server.jpg" alt="Сервер"/>
</a>
```

Списки

Список (нумерований або ненумерований) – блочний елемент, що містить всередині себе блочні елементи – члени списку.

Так записується в редакторі:

```
<ul>
<li>Перший рядок списку<li/>
<li>Другий рядок списку<li/>
<li>Третій рядок списку<li/>
</ul>
```

Так буде відображено:

- Перший рядок списку
- Другий рядок списку
- Третій рядок списку

Списки можуть бути нумерованими <ol> та ненумерованими <ul>. Також можливе вкладення списків один в один.

Виділення фрагментів тексту

Виділення відбувається за допомогою тегів , , <kbd>, <code>

Так записується в редакторі:

```
<p>Вивчаючи <code>HTML</code> слід звернути
<strong>особливу увагу</strong> на різницю між
<em>нумерованими</em> списками <kbd>&lt;ol&gt;
</kbd> і <em>ненумерованими</em> списками
<kbd>&lt;ul&gt;</kbd>.</p>
```

Так буде відображено:

Вивчаючи HTML слід звернути особливу увагу на різницю між нумерованими списками <ol> і ненумерованими списками <ul>.

Елементи заголовка HTML

```
<title>Це моя сторінка</title>
```

Відображується у заголовку сторінки в браузері.

```
<meta name="description" content="Важлива сторінка"/>
```

Описує вміст сторінки.

```
<meta name="keywords" content="мир, труд, май"/>
```

Описує ключові слова (часто використовується пошуковими машинами).

```
<meta name="author" content="Yevhen Berkunskyi"/>
```

Інформація про автора.

```
<meta http-equiv="Content-Language" content="ru">
```

Описує формат і кодування сторінки

```
<meta http-equiv="Content-Type" content="text/html;
charset=windows -1251">
```

Визначає основну мову, якою написано сторінку.

```
<meta  http-equiv="refresh"  content="5;
http://www.google.com/">
```

Описує частоту перезавантаження сторінки (в секундах) і, можливо, робить "redirection" іншу сторінку.

```
<link type="image/jpeg" rel="shortcut icon"
href="favicon.jpg">
```

Описує іконку, асоційовану зі сторінкою (застарілий варіант: помістити в корінний каталог сайту файл з іменем favicon.ico).

Лабораторна робота № 2.
Мова описання представлень CSS

Завдання. Використовуючи можливості CSS, змінити шаблон WEB-сайту, створений в Лабораторній роботі №1 у відповідності зі стандартом XHTML. Вимоги до шаблону:

- шаблон повинен проходити валідацію на ресурсі validator.w3.org;
- CSS файл не повинен містити стилів, коментарів, "хаків", які не використовуються в шаблоні;
- Верстка шаблона повинна бути семантично вірною.

Теоретичні відомості

Що таке CSS?

- Це мова описань зовнішнього представлення для вмісту, який описаний в HTML-сторінках;
- Визначає зовнішній вигляд тексту – шрифти, розміри, колір;
- Визначає розташування елементів відносно один одного;
- Описання зовнішнього представлення може бути фізично відокремлене від описання вмісту.

В колишньому стандарті HTML допускалось використання описань зовнішнього представлення за допомогою атрибутів та окремих елементів.

```
<p><fo,nt face="Arial">Ласкаво просимо в НУК ім.адм. Макарова. Ви отримаєте <b>найповнішу, <i>кращу, <u>НАЙКРАЩУ</u></i></b> освіту в Миколаєві з <font size="+1" color="red">ПОЧАТКОВИМ </font> багажом знань!</font></p>
```

Ласкаво просимо в НУК ім.адм.Макарова. Ви отримаєте ***найповнішу, кращу***, ***<u>НАЙКРАЩУ</u>*** освіту в Миколаєві з ПОЧАТКОВИМ багажом знань!

Проте такий спосіб описання представлення НЕ ПІДТРИМУЄТЬСЯ в "строгому" XHTML!

НЕПРАВИЛЬНО:

```
<p>
  <font face="Arial">В НУК ви отримаєте
    <b>найповнішу,
      <i>кращу,
        <u>якісну</u>
      </i>
    </b>освіту в Миколаєві з
```

```
      <font size="+1" color="red">початковим</font>
       багажом знань!
     </font>
   </p>
```

ПРАВИЛЬНО:

```
   <p style="font-family: Arial;">В НУК ви отри-
маєте
     <span style="font-width: bold;">найповнішу,
       <span style="font-style: italic;">кращу,
            <span style="text-decoration:
underline;">якісну</span>
       </span>
     </span>освіту в Миколаєві з
      <span style="font-size: larger; color:
red;">початковим</span>
     багажом знань!
   </p>
```

Прив'язування сторінки стилів до документу

Стилі можна вписати прямо в теги HTML-документа. Але частіше їх виносять або на початок документа, або в окремий файл.

Нехай файл *mystyles.css містить такий текст*

```
   p  { color: white; background-color: black; }
   h1 { font-size: large; font-weight: bold; }
   h2 { font-weight: 500; color: blue; }
```

Тоді сторінка HTML може містити посилання на нього у такому вигляді:

```
   <html>
     <head>
     <link rel="stylesheet" type="text/css"
     href=mystyles.css"/>
     </head>
     <body>
       вміст сторінки HTML-документа
     </body>
   </html>
```

Деякі атрибути та варіанти значень

Атрибути шрифту (font) і тексту (text).

```
font-family: "lucida console", "courier new", sans-
serif;
font-size: small;
font-size: larger;
font-size: 10px;
font-size: 80%;
font-weight: bold;
font-weight: 400;
font-style: italic;

font: sans-serif bold x-large;

text-align: center;
text-align: right;
text-transform: uppercase;
text-indent: 2cm;
text-decoration: underline;
text-decoration: blink;
```

Атрибути кольору

```
color: red;
color: rgb(25, 30, 120);
color: #c0c0c0;

background-color: yellow;
```

Взаємодія стилів

Якщо стиль визначений таким чином:

```
body { font-family: sans-serif; background-color:
yellow; }
p { color: red; background-color: aqua; }
a { text-decoration: overline underline; }
h2 { font-weight: bold; text-align: center; }
```

А сторінка, що підключає його, так:

```
<body>
  <h2>Це заголовок</h2>
  <p>А це абзац із
    <a href="myref.html">посиланням</a>
    всередині
```

```
  </p>
</body>
```

Тоді сторінка буде виглядати так:

Це заголовок

А це абзац із посиланням всередині

Використання тегів div та span

```
div.style1 { font-family: sans-serif; }
div.style2 { font-family: Times; color: blue; }
.bold { font-weight: bold; }
```

```
<body>
  <div class="style1">
    <h2>Це заголовок класа style1</h2>
    <p>Це абзац класу style1</p>
  </div>
  <div class="style2">
    <h2>Це заголовок классу style2</h2>
    <p>Це абзац <span class="bold">класу</span>
    style2</p>
  </div>
</body>
```

Це заголовок класу style1

Це абзац класу style1

Це заголовок класу style2

Це абзац **класу** style2

Каскадування стилів

Чим визначається стиль конкретного елемента?

1. Стилем, визначеним браузером "за замовчуванням".

2. Стилем, вказаним в окремій CSS-сторінці, що прив'язана до HTML-документу елементом <link> у заголовку:

```
<link rel="stylesheet" type="text/css"
href="styles.css"/>
```

3. Стилем, указаним в заголовку HTML-документа за допомогою елементу
<style>:

```
<style> body { background-color: yellow; } </style>
```

4. Стилем, указаним в самому елементі за допомогою атрибута style:
Відступ в половину дюйма від краю

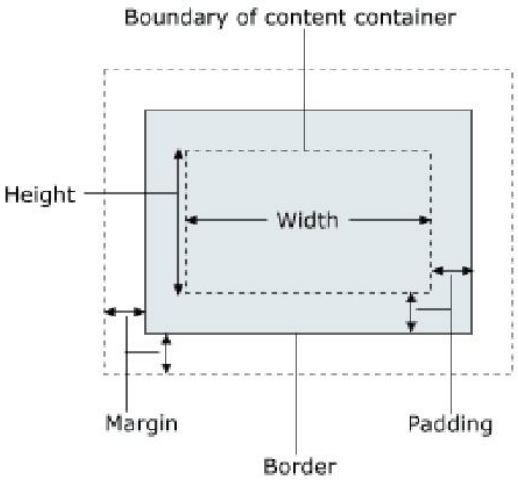
Чим "ближче" визначення стилю до елемента, тим приорітетнішим він буде у випадку конфлікту параметрів стилю.

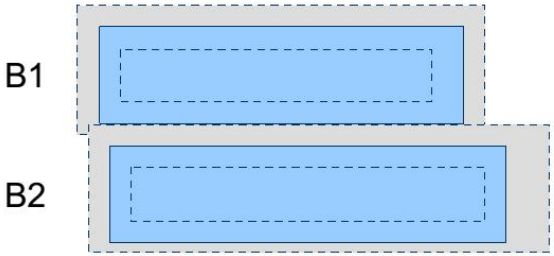
Розміщення фрагментів за допомогою CSS

Блочні елементи мають:

- внутрішній вміст вказаної ширини та висоти (width, height);
- прошарок (padding);
- межу (border);
- поля (margin).

За замовчуванням блоки розташовуються вертикально, при цьому поля сусідніх блоків перекриваються (спільне поле двох сусідніх блоків дорівнює по висоті максимальному з двох полів елементів).





Приклад розміщення блоків

```
<style type="text/css">
p { font-family: sans-serif;
    font-size: 16pt;
    border: 2px solid red; }
```

```
p.class1 { width: 400px;
           background-color: yellow;
           padding: 0.5cm;
           margin: 0.5cm; }
p.class2 { width: 500px;
           background-color: green;
           padding: 0.3cm;
           margin: 1cm; }
</style>
<body>
<p class="class1">Первый параграф</p>
<p class="class1">Второй параграф</p>
<p class="class2">Третий параграф</p>
<p class="class2">Четвертый параграф</p>
<p class="class1">Пятый параграф</p>
</body>
```

Розміщення блоків на сторінці

```
{ width: 70%; margin-left: auto; margin-right:
auto; }
```

Заданий таким чином, як показано вище, стиль дозволяє розмістити блок по центру сторінки (відповідно ліворуч чи праворуч, якщо задано одне поле).

Розміщення тексту (або інших рядкових елементів) всередині блока задається інакше:

```
{ text-align: left; }  (або center, або right)
<h1 style="width: 50%; margin-left: auto;
       background-color: yellow;">Заголовок</h1>
<h2 style="width: 70%; margin-left: auto;
      margin-right: auto; background-color: yellow;
      text-align: right;">ще один заголовок</h2>
```

Таким чином, комбінуючи можливості HTML та CSS, можна створити дизайн сторінок на будь-який смак, проте ці технології використовуються винятково на стороні клієнта. Якщо ж треба організувати взаємодію з сервером, треба розробляти відповідне серверне програмне забезпечення.

Лабораторна робота № 3.
Сервлети – серверні компоненти Java

Завдання. Розробити Web-застосування на основі технології сервлетів.

1. У середовищі NetBeans створити новий проект Web-застосування. Додати до цього проекту новий клас.

2. У створеному класі описати метод, що обчислює значення функції, яка задана у таблиці.

3. Розробити метод, що за вказаними значеннями кроку, початку та кінця інтервалу обчислює кількість кроків для табулювання.

4. Створити методи, що створюють масиви значень функції (*y*) та її аргументу (*x*) в усіх точках вказаного інтервалу із заданим кроком (розмір масивів обчислити програмно за допомогою метода з п.3). Масиви повинні бути описані як private і для них потрібно створити методи доступу до їхніх елементів за номерами.

5. Створити методи, що після формування масивів знаходять номери найбільшого та найменшого елементів масиву значень функції.

6. Вивести найбільший та найменший елементи масиву значень функції, вказавши їхні номери і відповідні значення аргументу.

7. Створити методи, що обчислюють і друкують суму та середнє арифметичне елементів масиву значень функції.

8. Додати до створеного проекту стартову HTML-сторінку для введення початкових даних, діапазону та кроку зміни аргумента.

9. Додати до проекту сервлет, що опрацьовує отримані дані: зчитує їх, передає створеному у пп. 2–7 класу, і відображає результати обчислень і пропонує ввести нові значення початкових даних.

10. Скомпілювати і виконати цю програму.

Теоретичні відомості

Сервлет – це застосування Java, що запускається і виконується в контейнері сервера застосувань. Клієнт спілкується з таким застосуванням за допомогою веб-браузера. Ніяких додаткових застосувань на стороні клієнта встановлювати не треба. Сервлети підтримуються віртуальною машиною JVM, що запобігати витоку пам'яті і забезпечити функціонування збирання сміття. Кожному клієнту сервлет виділяє незалежний потік виконання. Клієнт посилає застосуванню HTTP-запит, сервлет генерує відповідь і повертає її клієнту у вигляді html-документа.

Сервлет:

– Компонент застосувань Java Enterprise Edition;
– Завантажується веб-сервером в контейнер;
– Виконується на стороні сервера;
– Обробляє клієнтські запити;
– Динамічно генерує відповіді на запити;
– Знаходиться в стані очікування, якщо запити відсутні;
– Приймає запити від інших сервлетів (Servlet chaining);
– Підтримує з'єднання з ресурсами.

Найбільшого поширення набули сервлети, що обробляють клієнтські запити за протоколом HTTP. Технологія сервлетів є оболонкою протоколу HTTP і підтримує його як транспорт передачі даних від клієнта до сервера і назад. Контейнер сервлетов підтримує також протокол HTTPS (HTTP та SSL) для захищених запитів.

Для обробки HTTP-запитів у web можна скористатися в якості базового класу абстрактним класом HttpServlet з пакету javax.servlet.http.

Життєвий цикл сервлета починається з його ініціалізації та завантаження в пам'ять контейнером сервлетів при старті контейнера або у відповідь на перший клієнтський запит. Сервлет готовий до обслуговування будь-якого числа запитів. Завершення існування відбувається при вивантаженні його з контейнера.

Першим викликається метод **init()**. Він дає сервлету можливість ініціалізувати дані і підготуватися для обробки запитів. Найчастіше в цьому методі розміщується код, кешуючий дані фази ініціалізації.

Після цього сервлет можна вважати запущеним, він знаходиться в очікуванні запитів від клієнтів. Всі запити, що з'являються, обслуговуються методом **service (HttpServletRequest request, HttpServletResponse response)** сервлета, він викликається контейнером, а всі параметри запиту упаковуються в екземпляр request інтерфейсу **HttpServletRequest**, переданий в сервлет. Ще одним параметром цього методу є екземпляр **response** інтерфейсу **HttpServletResponse**, в який завантажується інформація для передачі клієнту. Для кожного нового клієнта при зверненні до сервлету створюється незалежний потік, в якому здійснюється виклик методу **service()**. Метод service() призначений для одночасної обробки безлічі запитів.

При вивантаженні додатка з контейнера, тобто після закінчення життєвого циклу сервлета, викликається метод **destroy(),** в тілі якого слід розміщувати код звільнення зайнятих сервлетом ресурсів.

Сервлети підтримуються серверами застосувань WebSphere, Weblogic, JBoss, Oracle OC4J Server, Glassfish, Geronimo, Apache Tomcat, а також контейнерами сервлетів tomcat, jetty, grizzly і є частиною платформи JEE.

При розробці сервлетів, як суперклас, в більшості випадків використовується не інтерфейс **Servlet**, а абстрактний клас **HttpServlet**, що відповідає за обробку запитів HTTP.

Метод **service()** класу **HttpServlet** служить диспетчером для інших методів, кожен з яких обробляє методи доступу до відповідних ресурсів. У специфікації HTTP визначені методи: GET, HEAD, POST, PUT, DELETE, OPTIONS і TRACE. Найбільш часто вживаються методи GET і POST, які передають на сервер запити, а також параметри для їх виконання.

Метод GET (method = "GET") використовується для запиту вмісту зазначеного ресурсу, зображення або гіпертекстового документа. Разом із запитом можуть передаватися додаткові параметри як частина URI, значення можуть вибиратися з полів форми або передаватися безпосередньо через URL.

При цьому запити кешуються і мають обмеження на розмір. цей метод є основним методом взаємодії браузера клієнта і веб-сервера.

Метод POST використовується для передачі даних користувача у вмісті HTTP-запиту на сервер. Користувацькі дані упаковані в тіло запиту згідно полю заголовка Content-Type і / або включені в URI запиту.

При використанні методу POST під URI мається на увазі ресурс, який буде обробляти запит.

Метод PUT схожий з методом POST за тим винятком, що тут URI означає ресурс, який буде створений або збережений на сервері в результаті виконання PUT-запиту.

Метод DELETE призначений для видалення цільового ресурсу. Обидві ці дії на деяких серверах можуть заборонятися через загрозу внутрішньої безпеки.

Метод HEAD передбачає повернення сервером такої ж відповіді, як і при використанні GET, але без тіла відповіді. Метод зазвичай використовується для того, щоб перевірити існування ресурсу або дізнатися, чи змінився ресурс з моменту останнього звернення.

Метод OPTIONS повинен повертати інформацію про можливості вебсервера або параметрах з'єднання для конкретного ресурсу.

Метод TRACE повертає клієнту запит у тому вигляді, в якому він прийшов на сервер – використовується для налагодження, визначаючи заголовки, що додаються проміжними серверами, а також для тестування налаштувань з'єднання.

Методи HEAD, OPTIONS і TRACE, як правило, реалізуються сервером веб-застосувань прозоро для програміста і не вимагають будь-якої спеціальної обробки. Тому зазвичай при розробці веб-застосувань вони не розглядаються.

У завдання методу **service()** класу HttpServlet входить аналіз отриманого через запит методу доступу до ресурсів і виклик методу з подібним ім'ям, де перед іменем додається префікс do: doGet (), doPost (), doHead (), doPut (), doDelete (), doOptions () і doTrace ().

Розробник повинен перевизначити потрібний метод, розмістивши в ньому функціональну логіку.

Механізмами протоколу HTTP не передбачено збереження інформації про свого клієнта. Однак веб-застосування може вимагати інформації про клієнта, особливо таке, реалізація якого передбачає аутентифікацію клієнта без необхідності її підтвердження при зміні сторінок. Крім підтримки розподіленої сесії, існують і примітивні механізми отримання відомостей про клієнта:

– файли cookie – текстові файли, асоційовані із застосуванням та зберігаються на комп'ютері клієнта;

– додавання в рядок GET-запиту додаткових параметрів;

– приховані HTML-теги вигляду

<input type = "hidden" name = "userType" value = "admin">

які необхідно додавати в кожну сторінку програми для збереження цілісності її архітектури.

Приклад сервлету.

```
package com.homelinux.berkut.web;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.annotation.WebServlet;
@WebServlet("/FirstServlettest")
public class FirstServlet extends HttpServlet {
   public FirstServlet() {
```

```
        super();
    }
    public void init() throws ServletException {
    }
    protected void doGet(
        HttpServletRequest request,
        HttpServletResponse response)
            throws ServletException, IOException {
      response.setContentType("text/html");
      response.getWriter().print(
          "This is " + this.getClass().getName()
           + ", using the GET method");
    }
    protected void doPost(
        HttpServletRequest request,
        HttpServletResponse response)
            throws ServletException, IOException {
      response.setContentType("text/html");
      response.getWriter()(
          "This is " + this.getClass().getName()
           + ", using the POST method");
    }
    public void destroy() {
     super.destroy(); // Just puts "destroy" string
in log
    }
  }
```

Починаючи з версії Servlet API 3.0, реєстрація сервлетів у файлі-дескрипторі web.xml необов'язкова. Конфігурація визначається за допомогою анотації WebServlet в розширеному вигляді із зазначенням імені сервлета в контексті:

```
@WebServlet(name = "FirstServletname", urlPatterns
= {"/FirstServlettest"})
public class FirstServlet extends HttpServlet {
}
```

Практика включення HTML-коду в код сервлета не вважається хорошою, оскільки ці дії "уводять" сервлет від його основної ролі – контролера застосування. Це призводить до зростання обсягу коду сервлета, який на певному етапі стає неконтрольованим і реалізує внаслідок цього антишаблон "Чарівний сервлет". Наведений вище сервлет має ознаки антишаблону, оскільки містить метод print() для формування коду HTML.

Сервлет повинен використовуватися тільки для контролю реалізації бізнес-логіки застосування і зобов'язаний бути відділений як від безпосереднього формування тексту відповіді на запит, так і від даних, необхідних для цього. Зазвичай для формування відповіді на запит застосовуються можливості JSP, JSPX, JSF і XHTML, але це – є темою наступної лабораторної роботи.

Лабораторна робота № 4.
JSP – серверні компоненти Java

Технологія Java Server Pages (JSP) забезпечує поділ динамічної і статичної частин сторінки, результатом чого є можливість зміни дизайну сторінки, не зачіпаючи динамічний зміст. Така властивість використовується при розробці та підтримці сторінок, оскільки дизайнерам немає необхідності знати, як працювати з динамічними даними.

JSP-код складається зі спеціальних тегів і виразів, які вказують контейнеру відповідність між тегами і java-кодом для генерації сервлету або його частини. Таким чином підтримується документ, який одночасно містить і статичну сторінку, і теги Java, які управляють сторінкою.

Статичні частини HTML-сторінок надсилаються у вигляді рядків в метод write(). Динамічні частини включаються прямо в код сервлету. З моменту формування відповіді сервера сторінка поводиться як звичайна HTML-сторінка з асоційованим сервлетом.

Щоб створити найпростішу JSP, досить взяти HTML-сторінку і замінити розширення html на jsp. Тільки в цьому випадку для запуску сторінки необхідний спеціальний application server з контейнером сервлетів і особливе розміщення самої сторінки.

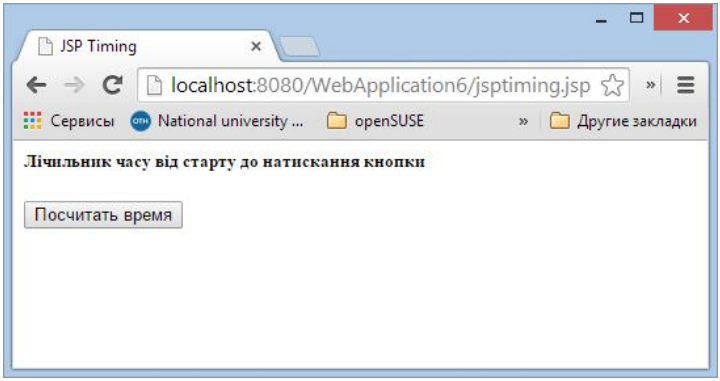
Взаємодія сервлета та JSP

Сторінки JSP і сервлети ніколи не слід використовувати в інформаційних системах один без одного. Причиною є принципово різні ролі, які грають дані компоненти в додатку. Сторінка JSP відповідальна за формування користувальницького інтерфейсу і відображення інформації, переданої з сервера. Сервлет виконує роль контролера запитів і відповідей, тобто приймає запити від всіх пов'язаних з ним JSP-сторінок, викликає відповідну бізнес-логіку для їх (запитів) обробки і залежно від результату виконання вирішує, яку JSP поставити цьому результату у відповідність.

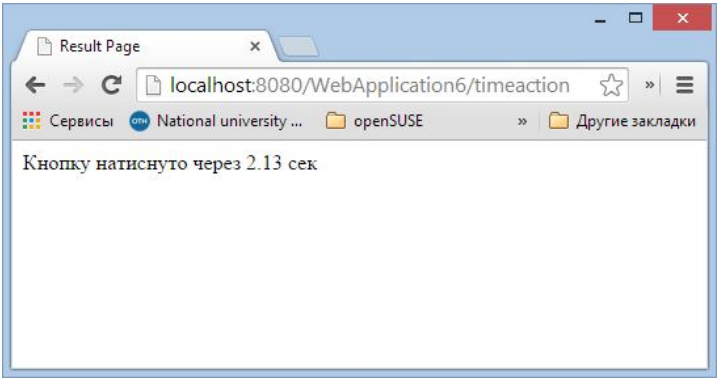
При необхідності розширення функціональності системи не слід створювати додаткові сервлети. Сервлет у застосуванні повинен бути один. При створенні нового навчального застосування, щоб уникнути плутанини завжди треба створювати новий проект.

Для демонстрації взаємодії JSP-сторінок і сервлету буде вирішена задача визначення часу між завантаженням сторінки в браузер і натисненням кнопки на цій же сторінці.

Сторінка JSP з наступним викликом інший JSP-сторінки, що відображує результати виконання запиту:




```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html><head><title>JSP Timing</title>
</head>
<body>
  <h5>Лічильник часу від старту до натискання кноп-
ки</h5>
<jsp:useBean                    id="calendar"
class="java.util.GregorianCalendar"/>
<form  name="Simple"  action="timeaction"
method="POST">
<input      type="hidden"      name="time"
value="${calendar.timeInMillis}"/>
<input type="submit" name="button" value="Посчитать
время"/>
</form>
</body></html>
```



```
<%@page contentType="text/html" pageEncoding="UTF-
8"%>
<html><head><title>Result Page</title></head><body>
<p>Кнопку натиснуто через ${res} сек </p>
</body></html>
```

Для того, щоб обробити запит, що надійде від першої сторінки, та сформувати дані для відображення на другій, потрібний сервлет:

```
package server;
import java.io.IOException;
import java.util.GregorianCalendar;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
@WebServlet(name = "TimeServlet", urlPatterns = {"/
timeaction"})
public class TimeServlet extends HttpServlet {
   protected void processRequest(HttpServletRequest
request,
                     HttpServletResponse response)
           throws ServletException, IOException {
GregorianCalendar gc = new GregorianCalendar();
String timeJsp = request.getParameter("time");
double delta = ((double)(gc.getTimeInMillis() -
                  Long.parseLong(timeJsp)))/1_000;
   request.setAttribute("res", delta);
   request.getRequestDispatcher(
        "/result.jsp").forward(request, response);
}
   @Override
   protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
            throws ServletException, IOException {
   processRequest(request, response);
}
   @Override
   protected void doPost(HttpServletRequest request,
                     HttpServletResponse response)
           throws ServletException, IOException {
   processRequest(request, response);
}
}
```

Висновок

У першій частині курсу було розглянуто лише основні можливості мов HTML та CSS для створення зовнішнього представлення Web-сторінок і базові можливості платформи Java для створення динамічного контенту та організації взаємодії з користувачем.

У другій частині курсу будуть розглянуті такі можливості web-технологій, як інтеграція з базами даних, web-сервіси та використання фреймворків для створення великих сайтів – порталів.

ЗМІСТ

Вступ 3
Лабораторна робота № 1. Мова розмітки HTML 4
Лабораторна робота № 2. Мова описання представлень CSS 9
Лабораторна робота № 3. Сервлети – серверні компоненти Java 15
Лабораторна робота № 4. JSP – серверні компоненти Java 21
Висновок 24